Laboratorul 07.

Arhiva laborator: Arhiva

Problema 1

Să se scrie un program pentru afișarea cuvintelor distincte dintr-un fișier text folosind clasa ***TreeSet***. Fișierul va conține și cuvinte repetate (identice). Elementele mulțimii se vor afișa în ordine crescătoare (metoda ***printWords***) și în ordine descrescătoare (metoda ***printWordsComparator***), fără a apela o funcție de sortare, folosind două obiecte ***TreeSet*** construite diferit (cu și fără argument de tip ***Comparator***). Se vor implementa cele 2 metode ce vor fi apelate din metoda ***main***:

- `public TreeSet printWords (String)` - va primi, ca argument, numele fișierului din care se citește și va folosi un obiect de tip ***TreeSet***, instanțiat folosind constructorul fără argument de tip ***Comparator***;
- `public TreeSet printWordsComparator(TreeSet)` - va primi ca argument obiectul de tip ***TreeSet***, returnat de metoda anterioară, și va construi si returna un alt obiect de tip ***TreeSet***, stocând informațiile în ordine descrescătoare.

În cazul instanțierii obiectului ce va fi returnat, folosiți constructorul cu argument de tip ***Comparator***!

Se poate folosi ca fisier de intrare fisierul test01.txt din arhiva!

Problema 2

Să se definească o clasă ***LinkedSet*** pentru o mulțime realizată ca listă înlănțuită de elemente distincte. Clasa va extinde clasa ***LinkedList*** și va implementa interfața ***Set***.

Se vor redefini cele două metode de adăugare și metoda de setare a unui obiect:

```
boolean add(Object obj);
void add (int i, Object obj);
Object set (int i , Object obj);
```

Să se testeze clasa prin folosirea metodelor definite și afișare! Se poate folosi Test2.java din arhiva!

Problema 3

Să se definească o clasă ***SListSet*** pentru o mulțime ordonată de obiecte, care extinde clasa ***LinkedList*** si implementează interfața ***SortedSet***. Clasa va conține un obiect de tip ***Comparator*** și metodele:

```
Comparator comparator () ; //comparator folosit (null pentru  comparatia naturala)
boolean add(Object o); //adauga un elemnt in multime daca nu exista deja si sorteaza multimea
Object first(); //primul obiect din multime
Object last(); // ultimul obiect din multime
SortedSet subSet(Object from, Object to); // o submultime ordonata
SortedSet headSet(Object to); // o submultime cu primele  obiecte
SortedSet tailSet(Object from); //o submultime cu ultimele obiecte
```

Se vor defini cel puțin doi constructori: fără argumente (sortare conform ordinii naturale) și cu un argument de tip ***Comparator***. Să se folosească un obiect de tip ***SListSet*** pentru afișarea cuvintelor distincte dintr-un text citit dintr-un fișier text (*test01.txt*), în ordine crescătoare sau descrescătoare (în locul clasei ***TreeSet*** din problema 1).

Problema 4

Program pentru afișarea numerelor liniilor dintr-un fișier text în care apare fiecare cuvânt distinct. Se va folosi un dicționar cu liste de valori asociate fiecărei chei. Dicționarul va fi de tip ***TreeMap***, iar listele vor fi de tip ***LinkedList***.

Pentru afisare se va folosi un Iterator pe multimea intrarilor din dictionar. La afișare, fiecare cuvânt va începe pe o linie nouă și va fi urmat, pe liniile următoare, de lista numerelor liniilor în care apare. Pentru fiecare cuvânt în parte se va afișa la sfârșit numărul de apariții al acestuia.

Se poate folosi ca fisier de intrare tot fisierul test01.in din arhiva!

Map - ***entrySet()***

Set - ***iterator()***

Problema 5

Program pentru crearea și afișarea unui dicționar cu numele fișierelor dintr-un director împreună cu dimensiunea lor exprimată în kiloocteți (ca număr întreg). Numele reprezintă cheia, iar dimensiunea este valoarea asociată cheii. Programul va folosi succesiv clasele ***HashMap*** și ***TreeMap***. Afișarea se va face atât în ordinea alfabetică a numelor cât și în ordinea dimensiunii fișierelor (două afișări).

`entrySet` din `Map` transformă colecția în `Set` de intrari;

`Collections.sort` sortează o colectie după un criteriu definit.